AMENDMENTS TO THE SPECIFICATION:

Please amend the specification as follows:

Please replace paragraph [0015] on page 7, with the following amended paragraph:

In accordance with a second aspect of the present invention, an interrupt process control method for performing data processing on a plurality of [[OS's]] OSs, includes steps of receiving, from a sub OS other than [[an]] a main OS, status information as to whether the sub OS is in an interrupt-enabled state or an interrupt-disabled state, detecting the generation of an interrupt, and controlling an interrupt process to perform one of an interrupt process execution and an interrupt process reserve, in response to the generation of the interrupt based on the status information.

Please replace paragraph [0016] on page 7, with the following amended paragraph:

In the interrupt process control method of one embodiment of the present invention, the main OS stores interrupt process status information as to whether the interrupt process is in progress or in reserve, and resumes the interrupt process execution in response to the transition of the sub OS between the interrupt-enabled state and the interrupt-disabled state.

Please replace paragraph [0017] on pages 7-8, with the following amended paragraph:

The interrupt process control method of one embodiment of the present invention may further include steps of notifying the main OS of the status information as to whether the sub OS is in the interrupt-enabled state or the interrupt-disabled state, and updating the status information of the sub OS in response to the notification from the sub OS.

Please replace paragraph [0019] on pages 8-9, with the following amended paragraph:

In the interrupt process control method of one embodiment of the present invention, the main OS performs status management based on a status table containing the status information of the sub OS and the interrupt process status information as to whether the interrupt process is in progress or in reserve. If an interrupt intended for the sub OS is generated and the main OS determines based on the status table that the sub OS is in the interrupt-disabled state, the main OS registers the interrupt in the status table as a reserved interrupt. If an interrupt intended for the sub OS is generated and the main OS determines based on the status table that the sub OS is in the interrupt-enabled state, the main OS performs interrupt control depending on whether the OS operating on a processor is either the main OS or the sub OS in a manner such that

(a) if the main OS is in operation, the main OS

(a1) executes the interrupt process in response to a high priority interrupt,

or

(a2) reserves the interrupt process in response to a low priority interrupt, and that

(b) if the sub OS is in operation, the sub OS executes the interrupt process regardless of the priority level of the interrupt.

Please replace paragraph [0021] on page 10, with the following amended paragraph:

In accordance with a third aspect of the present invention, a computer program for performing an interrupt process control in data processing on a plurality of [[OS's]] OSs, includes steps of receiving, from a sub OS other than [[an]] a main OS, status information as to whether the sub OS is in an interrupt-enabled state or an interrupt-disabled state, detecting the generation of an interrupt, and controlling an interrupt process to perform one of an interrupt process execution and an interrupt process reserve, in response to the generation of the interrupt based on the status information.

Please replace paragraph [0022] on page 10, with the following amended paragraph:

The computer program of one embodiment of the present invention is provided, to a general-purpose computer system executing a variety of program code, in a computer-readable storage medium, such as a CO, an FO, or an MO, or a communication medium such as network. By providing the computer program in a computer readable manner, the computer system performs ~~process~~ processes responsive to the computer program.

Please replace paragraph [0024] on page 11, with the following amended paragraph:

In accordance with embodiments of the present invention, the main OS executing the interrupt process is set in the system in which a plurality of [[OS's]] OSs concurrently run. With the main OS performing interrupt control ~~process~~ processes, the mask time of the entire system is reduced, interrupt response is improved, and data processing is efficiently performed.

Please replace paragraph [0025] on pages 11-12, with the following amended paragraph:

In accordance with embodiments of the present invention, the main OS executing the interrupt process control is set, and the right to set an interrupt mask is not handed over to [[the]] a sub OS other than the main OS. The sub OS notifies the main OS whether the sub OS is in the interrupt-enabled state or the interrupt-disabled state. The main OS controls an interrupt mask of the sub OS. This arrangement prevents the sub OS from reserving a required interrupt process due to its own mask control. In accordance with the intention of the main OS, all interrupt ~~process is~~ processes are controlled. Required interrupt processes are thus performed with priority.

Please replace paragraph [0026] on page 12, with the following amended paragraph:

In accordance with embodiments of the present invention, a sub OS interrupt vector management unit is set in the main OS. The main OS generally manages an interrupt vector area of the sub OS. Unlike interrupt vector management individually performed by [[OS's]] OSs, the interrupt vector is shared, and memory area is saved.

Please replace paragraph [0027] on pages 12-13, with the following amended paragraph:

[Fig. 1] Fig. 1 is a block diagram illustrating an information processing apparatus of one embodiment of the present invention.

[Fig. 2] Fig. 2 illustrates the structure of a processor module.

[Fig. 3] Fig. 3 illustrates operating systems of the information processing apparatus in accordance with one embodiment of the present invention.

[Fig. 4] Fig. 4 illustrates an allocation process of allocating a logical processor to a physical processor in a time sharing manner.

[Fig. 5] Fig. 5 is a functional diagram illustrating a main OS of the information processing apparatus in accordance with one embodiment of the present invention.

[Fig. 6] Fig. 6 illustrates a status table listing OS status information and interrupt process status information managed by the main OS in one embodiment of the present invention.

[Fig. 7] Fig. 7 illustrates the status of the sub OS and a process sequence responsive to the generation of an interrupt.

[Fig. 8] Fig. 8 illustrates a list of relationships of [[OS's]] OSs executing a process using a processor ([[OS's]] OSs in operation), priority level of the interrupt (high and low priority levels), an interrupt destination OS, and an interrupt grant state of the interrupt destination OS.

[Fig. 9] Fig. 9 illustrates a functional diagram depicting the notification of status information and the updating of status information according to an embodiment of the invention.

[Fig. 10] Fig. 10 illustrates a functional diagram that depicts interrupt process management based on status information of the sub operating system according to an embodiment of the invention.

[Fig. 11] Fig. 11 illustrates a functional diagram that depicts interrupt process management based on the priority of an interrupt and the operational status of the main and sub operating systems, according to an embodiment of the invention.

Please replace paragraph [0029] on pages 13-14, with the following amended paragraph:

The hardware structure of the information processing apparatus of one embodiment of the present invention is described below with reference to Fig. 1. A processor module 101 includes a plurality of processing units, and processes data in accordance with a variety of programs stored in a read-only memory (ROM) 104 and an HOD 123, including operating systems ([[OS's]] OSs) and application programs running on the [[OS]] OSs. The processor module 101 will be described later with reference to Fig. 2.

Please replace paragraph [0030] on page 14, with the following amended paragraph:

In response to a command input via the processor module 101, a graphic engine 102 generates data to be displayed on a screen of a display forming an output unit 122, and ~~for example,~~ performs, for example, a 3D graphic drawing process. A main memory (DRAM) 103 stores the program executed by the processor module 101 and parameters that vary in the course of execution of the program. These elements are interconnected via a host bus III including a CPU bus.

Please replace paragraph [0037] on page 16, with the following amended paragraph:

The structure of the processor module is described. The memory-flow controller in each processor group controls data inputting and data outputting to the main memory 103 of Fig. 1. The secondary cache serves as a memory area for ~~process~~ processing data in each processor group.

Please replace paragraph [0038] on page 16, with the following amended paragraph:

The operating systems ([[OS's]] OSs) of the information processing apparatus of one embodiment of the present invention are described below with reference to Fig. 3. The multi-OS information processing apparatus has a plurality of [[OS's]] OSs arranged in a logical layered structure as shown in Fig. 3.

Please replace paragraph [0039] on pages 16-17, with the following amended paragraph:

As shown in Fig. 3, a main OS 301 is arranged at a lower layer. A plurality of sub [[OS's]] OSs 302, 303, and 304 are arranged at upper layers. Guest sub [[OS's]] OSs 302 and 303 and a sub [[a]] system control OS 304 are set as sub [[OS's]] OSs. Together with the system control OS 304, the main OS 301 forms a logical partition as an execution unit of each process executed by the processor module 101 discussed with reference to Figs. 1 and 2, and allocates system hardware resources (for example, main processors, sub-processors, memories, and devices, as computing resources) to each logical partition.

Please replace paragraph [0040] on page 17, with the following amended paragraph:

The guest [[OS's]] OSs 302 and 303 as the sub [[OS's]] OSs are a gaming OS, Windows®, Linux®, etc, and operate under the control of the main OS 301. Although only two guest [[OS's]] OSs 302 and 303 are shown in Fig. 3, the number of guest [[OS's]] OSs is not limited to two.

Please replace paragraph [0041] on page 17, with the following amended paragraph:

The guest [[OS's]] OSs 302 and 303 operate within the logical partitions set by the main OS 301 and the system control OS 304. The guest [[OS's]] OSs 302 and 303

process a variety of data using hardware resources such as main processors, sub-processors, memories, and ~~device~~ devices, each allocated to the logical partition.

Please replace paragraph [0043] on page 18, with the following amended paragraph:

The system control as 304 as one of the sub [[OS's]] OSs generates a system control program 307 containing a logical partition management program, and performs operation control responsive to the system control program 307 together with the main OS 301. The system control program 307 controls system policy using a system control program programming interface. The application program 306 is supplied with the system control program programming interface by the control OS 301. For example, the system control program 307 permits flexible customization, for example, setting an upper limit on resource allocation.

Please replace paragraph [0044] on pages 18-19, with the following amended paragraph:

The system control program 307 controls the behavior of the system using the system control program programming interface. For example, the system control program 307 produces a new logical partition, and starts up a new guest OS at the logical partition. In a system where a plurality of guest [[OS's]] OSs are operating, the guest [[OS's]] OSs are initiated in the order programmed in the system control program 307. The system control program 307 can receive and examine a resource allocation request issued from the guest OS before being received by the main

OS 301, modify the system policy, and deny the request itself. In this way, no particular guest OS monopolizes the resources. A program into which the system policy is implemented is the system control program.

Please replace paragraph [0057] on page 24, with the following amended paragraph:

The information processing apparatus of one embodiment of the present invention is a multi-OS system using a plurality of [[OS's]] OSs. As previously discussed with reference to Fig. 3, the plurality of [[OS's]] OSs are divided into a single main OS and remaining sub [[OS's]] OSs. The function and interrupt process control of the main OS and the sub [[OS's]] OSs of the information processing apparatus of one embodiment of the present invention are described below.

Please replace paragraph [0059] on page 25, with the following amended paragraph:

The functions of in the interrupt process control of the main OS and the sub OS are listed as follows:

(1) The main OS manages competing resources such as setting an interrupt mask and interrupt vector.

(2) The main OS records status information of [[OS's]] OSs processing data, including the main OS.

(3) Instead of directly controlling an interrupt mask register, the sub OS notifies the main OS whether the sub OS is in an interrupt-enabled state or an interrupt-disabled

state.

(4) The main OS receives all interrupt requests, and transfers a required interrupt only when the sub OS is interruptable and operative.

Please replace paragraph [0064] on pages 26-27, with the following amended paragraph:

The sub OS interrupt manager 512 manages the interrupt-enabled state and the interrupt-disabled state of the sub OS. As previously discussed, the main OS 510 is notified by the sub OS 520 whether the sub OS is in the interrupt-enabled state or the interrupt-disabled state. In response to the notification from the sub OS 520, the sub OS interrupt manager 512 in the main OS 510 stores management information as to whether each sub OS is in the interrupt-enabled state or the interrupt-disabled state.

Please replace paragraph [0065] on page 27, with the following amended paragraph:

The sub OS interrupt manager 512 registers the state of each sub OS onto a status table listing the state as to whether each sub OS is in the interrupt-enabled state or the interrupt-disabled state. For example, the status table contains an interrupt mask register that stores the interrupt-enabled state as (0) and the interrupt-disabled state as (1). The sub OS interrupt manager 512 manages the state of each sub OS based on the interrupt mask register. If an interrupt request occurs with the sub OS in the interrupt-disabled state, namely, in a masked state, the interrupt request must wait. With the sub

OS in the ~~interrupt-disabled~~ interrupt-enabled state, namely, in a mask released state, the interrupt process responsive to the interrupt request is permitted.

Please replace paragraph [0066] on pages 27-28, with the following amended paragraph:

Fig. 5 illustrates a single sub OS 520. As previously discussed with reference to Fig. 3, the information processing apparatus can accommodate a plurality of sub [[OS's]] OSs. The sub OS interrupt manager 512 in the main OS 510 stores the management information of each of the sub [[OS's]] OSs as to whether the sub OS is in the interrupt-enabled state or the interrupt-disabled state, and performs interrupt process control in response to the state of each sub OS.

Please replace paragraph [0067] on page 28, with the following amended paragraph:

Fig. 6 illustrates the status table stored as the management information of the sub OS interrupt manager 512. As shown in Fig. 6, the status table lists, in a related form, data of the state of each OS, namely, the status information as to whether the sub OS is in the interrupt-enabled state or the interrupt-disabled state, information relating to a reserved interrupt, and information relating to the interrupt process in progress. Upon receiving the status notification from each sub OS, the sub OS interrupt manager 512 updates the status table in response to the received information. In response to the occurrence, reserve, delivery, and completion of an interrupt process, the sub OS

interrupt manager 512 registers in the status table the status as to whether each interrupt is reserved or in progress. The sub OS interrupt manager 512 deletes an interrupt from the status table if the corresponding interrupt process has been completed.

Please replace paragraph [0071] on page 30, with the following amended paragraph:

The executing OS switch controller 517 controls switching a variety of data ~~processings~~ processing (including tasks) to be performed by the processor. As previously discussed, the logical processors are allocated to the physical processor. At least one processor is shared in time by the [[OS's]] OSs to process data. The OS switch controller 517 switches the [[OS's]] OSs in accordance with the processor use scheduling of the [[OS's]] OSs.

Please replace paragraph [0073] on pages 30-31, with the following amended paragraph:

The interrupt reserve controller 519 performs an interrupt reserve control process. As previously discussed, the interrupt request occurring in the interrupt-disabled state, namely, in the masked state must wait until masking is released. The interrupt reserve controller 519 manages a reserved interrupt on standby state. The interrupt reserve controller 519 updates the status table having interrupt reserve information set therewithin and notifies the sub OS 520 of the reserved interrupt if the reserved interrupt is registered in the status table.

Please replace paragraph [0074] on page 31, with the following amended paragraph:

The interrupt process control and state transition performed by the main OS in response to the two states of the sub OS, namely, the "interrupt_enabled state" and the "interrupt_disabled state", is described below with reference to Fig. 7.

Please replace paragraph [0075] on page 31, with the following amended paragraph:

As shown in Fig. 7, steps S101 - S104 show fours states of a single sub OS in the information processing apparatus as follows:

State S101: the sub OS starts.

State S102: the sub OS is initialized.

State S103: the sub OS is set to be in the interrupt_disabled state.

State S104: the sub OS is set to be in the interrupt_enabled state.

Please replace paragraph [0076] on pages 31-32, with the following amended paragraph:

With the sub OS being in one of:

state S103: interrupt_disabled state and

state S104: interrupt_enabled state, one of the following events occurs:

event 1201: interrupt to the main OS, or

event 1202: interrupt to the sub OS.

Please replace paragraph [0079] on page 33, with the following amended paragraph:

[Sequence 2]

The sequence 2 is performed in response to the state transition from step S102: completion of initialization of the sub OS to step S103: interrupt-disabled state of the sub OS.

During the ~~transitional~~ transition, the sub OS notifies the main OS that the sub OS is in the interrupt-disabled state. This step is performed as the notification step to the sub OS interrupt manager 512 of Fig. 5.

Please replace paragraph [0080] on page 33, with the following amended paragraph:

Upon receiving the notification from the sub OS that the sub OS is in the interrupt-disabled state, the sub OS interrupt manager 512 of the main OS registers the interrupt-disabled state of the sub OS in the ~~state~~ status table listing the mask state (see Fig. 6). By controlling the interrupt mask register with the interrupt-disabled state set as being (0) and the interrupt-enabled state set as being (1), the sub OS interrupt manager 512 sets a (mask) state to indicate that the sub OS is in the interrupt-disabled state. If an interrupt request is input under this state, that interrupt request must wait.

Please replace paragraph [0081] on pages 33-34, with the following amended paragraph:

[Sequence 3]

The sequence 3 is performed in response to a state transition from state S102: completion of initialization of the sub OS to state S104: interrupt-enabled state of the sub OS.

During the state transition, the sub OS notifies the main OS that the sub OS is in the interrupt-enabled state. The step is performed as the notification step to the sub OS interrupt manager 512 of Fig. 5.

Please replace paragraph [0082] on page 34, with the following amended paragraph:

Upon receiving from the sub OS the notification that the sub OS is in the interrupt-enabled state, the sub OS interrupt manager 512 in the main OS registers the interrupt-enabled state of the sub OS in the status table listing the mask state (see Fig. 6). By controlling the interrupt mask register with the interrupt-disabled state set as being (0) and the interrupt-enabled state set as being (1), the sub OS interrupt manager 512 sets a (mask release) state to indicate that the sub OS is in the interrupt-enabled state. If an interrupt request is input under this state, that interrupt request is performed without waiting.

Please replace paragraph [0083] on pages 34-35, with the following amended paragraph:

The main OS checks the ~~state~~ status table for the presence of an interrupt currently reserved. The status table lists OS status information as to whether each OS

is in the interrupt-enabled state or the interrupt-disabled state, and interrupt reserved state information. The interrupt reserve controller 519 of Fig. 5 writes the interrupt reserved state information onto the status table. If a reserved interrupt is present, the interrupt reserve controller 519 notifies the sub OS of the reserved interrupt.

Please replace paragraph [0084] on page 35, with the following amended paragraph:

[Sequence 4]

The sequence 4 is performed in response to a state transition from

state S104: sub OS interrupt-enabled state to

state S103: sub OS interrupt-disabled state.

During the state transition, the sub OS notifies the main OS that the sub OS is transitioned from the interrupt-enabled state to the interrupt-disabled state. This step is performed as the notification step to the sub OS interrupt manager 512 of Fig. 5.

Please replace paragraph [0085] on pages 35-36, with the following amended paragraph:

Upon receiving from the sub OS the notification that the sub OS is transitioned from the interrupt-enabled state to the interrupt-disabled state, the sub OS interrupt manager 512 in the main OS updates the status table (see Fig. 6) listing the mask state, namely, changing state registration information of the sub OS from the interrupt-enabled state to the interrupt-disabled state. By controlling the interrupt mask register with the interrupt-disabled state set as being (0) and the interrupt-enabled state set as being (1),

the sub OS interrupt manager 512 sets a (mask) state to indicate that the sub OS is in the interrupt-disabled state. If an interrupt request is input under this state, that interrupt request must wait.

Please replace paragraph [0086] on page 36, with the following amended paragraph:

[Sequence 5]

The sequence 5 is performed in response to a state transition from

state S103: sub OS interrupt-disabled state to

state S104: sub OS interrupt-enabled state.

During the state transition, the sub OS notifies the main OS that the sub OS is transitioned from the interrupt-disabled state to the interrupt-enabled state. This step is performed as the notification step to the sub OS interrupt manager 512 of Fig. 5.

Please replace paragraph [0087] on pages 36-37, with the following amended paragraph:

Upon receiving from the sub OS the notification that the sub OS is transitioned from the interrupt-disabled state to the interrupt-enabled state, the sub OS interrupt manager 512 in the main OS updates the status table (see Fig. 6) listing the mask state, thereby changing the state registration information of the sub OS from the interrupt-disabled state to the interrupt-enabled state. By controlling the interrupt mask register with the interrupt-disabled state set as being (0) and the interrupt-enabled state set as being (1), the sub OS interrupt manager 512 sets a (mask released) state to indicate

that the sub OS is in the interrupt-enabled state. If an interrupt request is input under this state, the interrupt process is performed.

Please replace paragraph [0088] on page 37, with the following amended paragraph:

[Sequence 6]

The sequence 6 is performed when the sub OS interrupt occurs as the event I202 in the sub OS interrupt-disabled state of S103. Upon detecting the event I202, namely, the occurrence of the sub OS interrupt, the main OS references the status table to determine whether the sub OS is in the interrupt-enabled state or the interrupt-disabled state. The sub OS is now in the interrupt-disabled state.

After verifying that the sub OS is in the interrupt-disabled state, the main OS registers the interrupt as reserved on the status table.

The interrupt process is not currently performed but reserved.

Please replace paragraph [0089] on page 37, with the following amended paragraph:

[Sequence 7]

The sequence 7 is performed when the sub OS interrupt occurs as the event I202 in the sub OS interrupt-enabled state of S104.

Please replace paragraph [0090] on pages 37-38, with the following amended paragraph:

The process in the sequence 7 becomes different depending on whether the process of the processor is performed by the main OS or the sub OS.

(a) Main OS in operation

The main OS detects the sub OS interrupt as the event I202 while performing the process using the processor. The main OS references [[of]] the status table of the sub OS to determine whether the sub OS is in the interrupt-enabled state or the interrupt-disabled state. The sub OS is in the interrupt-enabled state (S104).

After verifying that the sub OS is in the interrupt-enabled state, the main OS examines the priority of the interrupt. The interrupt is prioritized with a high priority level or a low priority level. The interrupt process responsive to the high priority level and the low priority level is described below.

Please replace paragraph [0093] on pages 40-41, with the following amended paragraph:

(b) Sub OS in operation

If the sub OS performs the process using the processor, the following process steps are performed. In the following, [Main OS] represents a process performed by the main OS, and [Sub OS] represents a process performed by the sub OS.

(1) [Main OS] Upon detecting the generation of a sub OS interrupt, the main OS references the status table of the sub OS to determine whether the sub OS is in the interrupt-enabled state or the interrupt-disabled state. In this case, the sub OS is in the interrupt-enabled state (S104).

(2) [Main OS] After verifying that the sub OS is in the interrupt-enabled state, the main OS registers the generated interrupt, as being processed, as data of the sub OS corresponding to the interrupt in the status table (Fig. 6).

(3) [Main OS] The main OS delivers the interrupt request to the sub OS. This process step is performed as a step of the interrupt deliverer 516 of Fig. 5.

(4) [Sub OS] The sub OS performs the interrupt process.

(5) [Sub OS] Upon completing the interrupt process, the sub OS notifies the main OS of the end of the interrupt process. The interrupt process end notifier 518 of Fig. 5 receives the end notice from the sub OS.

(6) [Main OS] The main OS deletes from the status table an entry corresponding to the interrupt process registered and completed.

Please replace paragraph [0094] on page 41, with the following amended paragraph:

The sub OS performs the interrupt process responsive to the generated interrupt using the processor. The sub OS, if in the interrupt-enabled state, performs the interrupt process without waiting.

Please replace paragraph [0095] on pages 41-42, with the following amended paragraph:

In summary, the interrupt process is performed in the sequence 7, namely, when the sub OS interrupt occurs with the sub OS in the interrupt-enable state.

(A) If the main OS is in operation,

(A-1) the interrupt process is performed in response to a high priority level interrupt, and

(A-2) the interrupt process is reserved in response to a low priority level interrupt.

(B) If the sub OS is in operation, the interrupt process is performed regardless of the priority level of the interrupt.

Please replace paragraph [0096] on pages 42, with the following amended paragraph:

[Sequence 8]

The sequence 8 is performed if the event I201 as the main OS interrupt takes place with the sub OS in the interrupt-disabled state of S103.

Please replace paragraph [0101] on page 44, with the following amended paragraph:

[Sequence 9]

The sequence 9 is performed if the event I201 as a main OS interrupt takes place with the sub OS in the interrupt-enabled state of S104.

Please replace paragraph [0102] on page 44, with the following amended paragraph:

This process is identical to the process at the above-referenced sequence 8. More specifically, if an interrupt intended for the main OS takes place, the same process

is performed regardless of whether the sub OS is in the interrupt-enabled state or the interrupt-disabled state.

The processes performed in response to the main OS interrupt (sequences 8 and 9) are summarized as below.

(A) The interrupt process is performed regardless of the interrupt priority level if the interrupt occurs while the main OS is in operation.

(B) If an interrupt occurs while the sub OS is in operation, the interrupt process is

(B-1) performed in response to the high priority level interrupt but

(B-2) reserved in response to the low priority level interrupt.

Please replace paragraph [0103] on pages 44-45, with the following amended paragraph:

Fig. 8 lists [[OS's]] OSs executing the processor applied process ([[OS']] OSs in operation), the priority levels (high or low), interrupt destination [[OS's]] OSs, and interrupt-enabled and disabled states of the interrupt destination [[OS's]] OSs. As listed, 16 scenarios of state setting are possible.

Please replace paragraph [0104] on page 45, with the following amended paragraph:

In accordance with one embodiment of the present invention, the main OS stores and manages the status information (status table of Fig. 7) of all sub [[OS's]] OSs as to whether each sub OS is in the interrupt-enabled state or the interrupt-disabled state.

Please replace paragraph [0106] on pages 45-46, with the following amended paragraph:

[Response to the sub OS interrupt process]

When an interrupt intended for the sub OS takes place, the main OS references the status table. If the sub OS is in the interrupt-disabled state, the main OS sets and registers, in the status table, the generated interrupt as being reserved (sequence 6).

If the sub OS is in the interrupt-enabled state, the interrupt reserve process or the interrupt process is performed as below depending on whether the OS executing the processor applied process is the main OS or the sub OS (sequence 7).

(A) If the main OS is in operation, the interrupt process is

(A-l) executed in response to the high priority level interrupt but

(A-2) reserved in response to the low priority level interrupt.

(B) If the sub OS is in operation, the interrupt process is executed regardless of the priority level of the interrupt.

Please replace paragraph [0108] on pages 46-47, with the following amended paragraph:

The main OS controls execution of the interrupt process and the interrupt reserve process based on the status information of all sub [[OS's]] OSs, namely, depending on whether each sub OS is in the interrupt-enabled state or the interrupt-disabled state, status information at the generation of the interrupt, namely, the priority level of

the interrupt, and whether the generated interrupt is intended for the main OS or the sub OS.

Please replace paragraph [0109] on pages 47-48, with the following amended paragraph:

In accordance with embodiments of the present invention, the right to mask interrupts is not handed over to the sub OS. The sub OS notifies the main OS whether the sub OS is in the interrupt-enabled state and the interrupt-disabled state. Based on the notification, the main OS controls the interrupt masking to the sub OS. This arrangement overcomes the inconvenience that the sub OS reserves a required interrupt process in accordance with the sub OS's own mask control. All interrupt ~~process is~~ processes are thus controlled by the main OS. Since the sub OS interrupt vector manager 513 in the main OS (see Fig. 5) manages the interrupt vector area of the sub OS, the interrupt vector is shared, unlike the case in which interrupt vector management is performed by individual [[OS's]] OSs. As previously discussed, the interrupt vector is the table of the memory area defined by the interrupt factor. For example, the interrupt vector is composed of a start address of an interrupt process routine. A processor receiving an interrupt examines the address of an interrupt handler from the memory area, and jumps to the address to start the interrupt process. The interrupt vector is shared by the main OS and the sub OS, and the required memory area is thus reduced.

Please replace paragraph [0111] on page 48, with the following amended paragraph:

The ~~above-references~~ above-referenced series of steps can be performed by software, hardware, or a combination thereof. If the series of steps is performed by software, a program forming the software is installed from a recording medium or via a network onto a computer incorporated into a hardware structure or to a general-purpose computer performing a variety of processes, for example.

Please replace paragraph [0115] on page 50, with the following amended paragraph:

In accordance with embodiments of the present invention, the main operating system (OS) executing the interrupt process is set in the system in which a plurality of [[OS's]] OSs concurrently run. With the main OS performing the interrupt control process, the mask time of the entire system is reduced, interrupt response is improved, and data processing is efficiently performed.

Please replace paragraph [0116] on page 50, with the following amended paragraph:

In accordance with embodiments of the present invention, the main OS executing the interrupt process control is set, and the right to set an interrupt mask is not handed over to the sub OS other than the main OS. The sub OS notifies the main OS as to whether the sub OS is in the interrupt-enabled state or the interrupt-disabled state. The main OS controls an interrupt mask of the sub OS. This arrangement prevents the sub OS from reserving a required interrupt process due to its own mask control. In

accordance with the intention of the main OS, all interrupt ~~process is~~ processes are controlled. Required interrupt processes are thus performed with priority.

Please replace paragraph [0117] on pages 50-51, with the following amended paragraph:

In accordance with embodiments of the present invention, the sub OS interrupt vector management unit is set in the main OS. The main OS generally manages the interrupt vector area of the sub OS. Unlike interrupt vector management individually performed by [[OS's]] OSs, the interrupt vector is shared, and memory area is reduced.

Please add new paragraph [0118]:

Fig. 9 illustrates the notification of status information and the updating of status information performed by an information processing apparatus according to an embodiment of the invention. A sub OS (910) checks status information in step 930, and notifies the main OS (920) on the status of the sub OS. For example, the main OS is notified as to whether the status information indicates an interrupt-enabled or interrupt-disabled state. Following the notification, the main OS updates the status information of the sub OS in step 940.

Please add new paragraph [0119]:

Fig. 10 illustrates interrupt process management performed by an information processing apparatus according to an embodiment of the invention. In step 1010, a determination is made with regard to whether a generated interrupt is intended for a sub

operating system. If the interrupt is intended for a sub operating system, then step 1020 is invoked, where the status of the sub operating system is determined. If the status of the sub operating system indicates an interrupt-disabled state, then the interrupt is registered as a reserved interrupt in step 1030. If the status of the sub operating system indicates an interrupt-enabled state, then step 1040 is invoked, where it is determined whether the main operating system or the sub operating system is in operation. If the sub operating system is in operation, then the interrupt process is executed in step 1050. If the main operating system is in operation, then step 1060 is invoked, where the priority of the interrupt is determined. If the interrupt is a high-priority interrupt, then the interrupt process is executed in step 1080. On the other hand, if the interrupt is a low-priority interrupt, then the interrupt process is reserved in step 1070.

Please add new paragraph [0120]:

Fig. 11 illustrates interrupt process management performed by an information processing apparatus according to an embodiment of the invention. In step 1100, a determination is made with regard to whether a generated interrupt is intended for the main operating system. If the generated interrupt is intended for the main operation system, then at step 1110, it is determined whether the main operating system or a sub operating system is in operation. If the main operating system is in operation, then the interrupt process is executed in step 1120. If the sub operating system is in operation, then the priority of the interrupt is determined in step 1130. If the interrupt is of a low

priority, then the interrupt process is reserved in step 1150. If, on the other hand, the interrupt is of a high priority, the interrupt process is executed in step 1140.